ECE 485

Lab 2: Audio Effects and Real-Time Processing

Jason Fischell

Due: Monday, March 25, 2019

Duke University

Instructor: Dr. Pfister

I have adhered to the Duke Community Standard in completing this assignment.

Contents

1 Real-Time Audio Warm-Ups

1.1 Warm-Up 1: Generating an Audio Signal

1.1.1 Without a Buffer

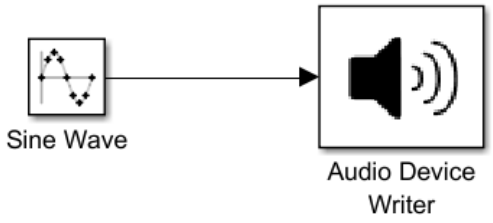

Figure 1: Sine wave audio output without a buffer

The image above, Figure 1, depicts the real time audio output of a sine wave without a buffer. This signal, when played, produces a tone filled with rapid clicking. This is likely the result of lag in the computer processing. After the computer plays a sound, there is some delay while the system is processing the next sound to play, and in this delay there is silence, hence the pauses between clicks.

1.1.2 Size 64 Buffer

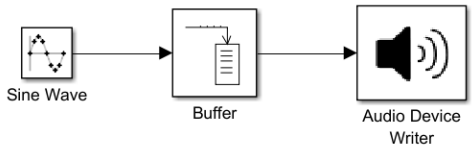

Figure 2: Sine wave audio output with a buffer

The image above, Figure 2, the addition of a buffer seemed to help, but not enough to fix the problem. The duration of tones became longer, but so did the duration of pauses. This is likely because the buffer is not large enough, and adds an extra step in processing, thereby increasing the delays between tones instead of eliminating them. This is buffer underflow, because the auido writer is requesting data faster than the computer can process it, and should be resolved by increasing the size of the buffer.

1.1.3 Size 1024 Buffer

By increasing the size of the buffer from 64 to 1024, the problem was eliminated, with the system outputting a consistent low buzz. No image was included because the system still appears identical to that depicted in Figure 2.

1.2 Warm-Up 2: Processing an Audio Signal

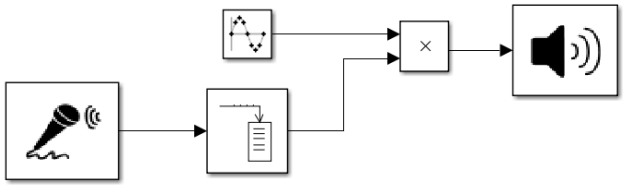

Figure 3: Microphone input with a sine wave envelope

The image above depicts a system that multiplies a microphone input by a sine wave envelope, causing the amplitude (volume) of the microphone recorded sound to rise and fall with the sine wave. This was difficult to detect in a real time system, so a delay was added before the audio output to make it more apparent. As depicted in the equalities below, the frequency of the sine wave envelope was calculated to be 0.538 Hz.

$$T = \frac{80 samples}{period} \cdot \frac{1024 seconds}{44100 samples} = 1.858 \frac{seconds}{period} \tag{1}$$

$$f = \frac{1}{T} = 0.5383 Hz \tag{2}$$

1.3 Warm-Up 3: Simple Echo

The simple echo effect was created as described in the laboratory manual. The delay with a gain of 0.5 was evident. By adjusting the output buffer size, the minimum output latency without buffer underflow was determined to occur around output buffer size 128 samples. At output buffer size 120 samples, underflow was obvious, and by 160 samples, there was no underflow. Buffer underflow was virtually undetectable (if not nonexistent) at an output buffer size of 128 samples.

1.4 Warm-Up 4: Filtering Noise in Audio Signal

In this exercise, a biquad filter was applied to a sinusoidal input. Based on the constant coefficients applied to the biquad filter, its transfer function $H(z)$ is as follows:

$$H(z) = \frac{1 - z}{1 - 0.9z} \tag{3}$$

Analysis of this transfer function yields a zero at DC (z = 1, $\omega = 0$) and pole, also on the real axis at z = $1.\overline{1}$. The zero at DC serves to make this is a high pass filter, and the nearby pole serves to cancel out the effect of the zero at high frequencies ($\omega >> 0$). The proximity of the pole and the zero should serve to make the gain roughly 1 at high frequencies. This theory was confirmed by testing the output of the filter at frequencies of 1 Hz, 50 Hz, 500 Hz, and 5000 Hz, as depicted in Figure 4 below.

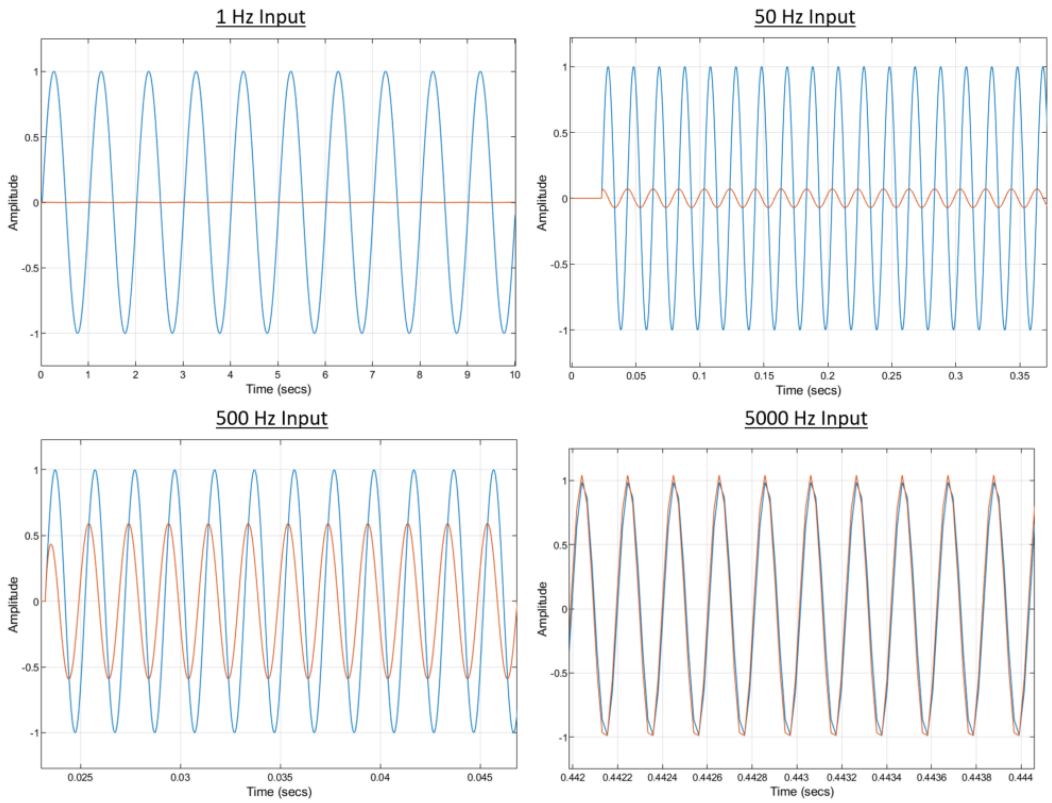

Figure 4: Output of the filter at various frequencies. These results confirm that this is a high pass filter. The 1 Hz signal was fully attenuated (top left) and the 5000 Hz signal was not attenuated at all (bottom right). The blue traces represent the input signal, and the orange lines represent the output.

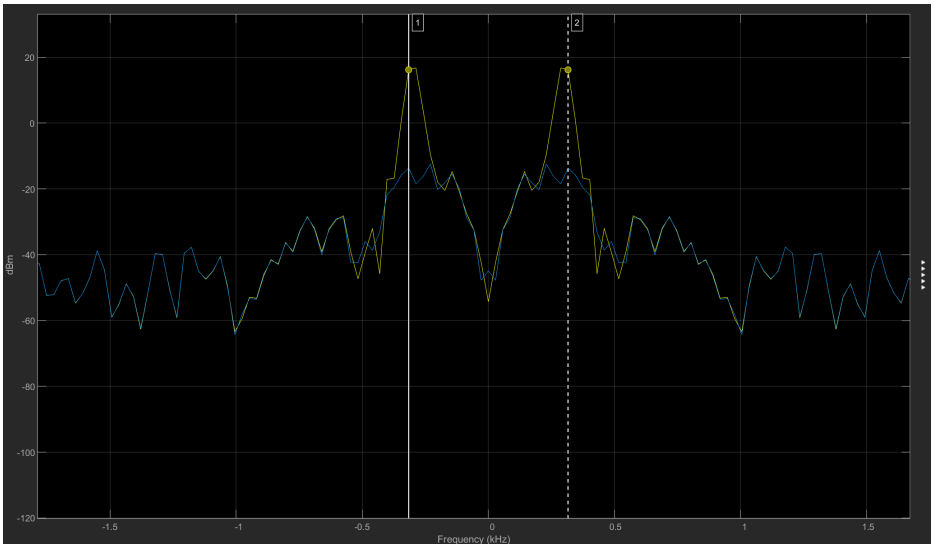

Figure 5: Output from the spectrum analyzer where yellow represents the sum of the 300 Hz sinusoid and guitar1.wav before filtering, and blue represents the sum after filtering. The peaks are the result of the 300 Hz sine noise added to the signal, which is why they occur at ($\pm$) 300 Hz.

1.5 Audio Plugins in Matlab

The pitch shifting audio plugin was used, and the results were very amusing.

2 Basic Effects

2.1 Exercise A: Wah

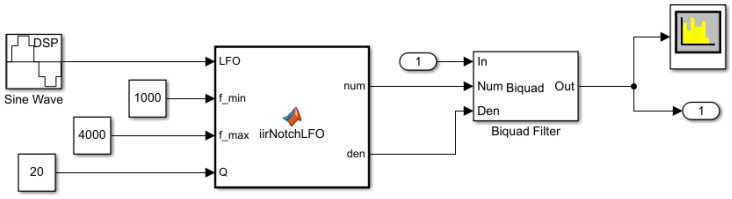

Figure 6: The system used to generate the wah effect, based on code described in the laboratory manual.

The effect that was generated in this section (see Figure 6) was wah. In other words, a band pass filter with a slowly oscillating center frequency was applied to the system. The results were as expected, with the frequencies within the band pass filter being accentuated as the center frequency changed. The frequency of oscillation was changed by changing the frequency of the input LFO. The upper and lower limits of the frequency oscillation are also set by constants as inputs to the filter block, as was the W factor, which determines the steepness of the band pass filter. The block of code in Section 4.1 outputs the numerator and denominator of a biquad filter as depicted in Figure 6.

The results of implementing this filter were quite satisfying. It was noticeably similar to the same effect applied in many songs that I know. Modifying the LFO frequency, upper and lower limits of the band pass center frequency, and the Q factor allowed me to tune the effect to best suit the sound and my computer speakers. After this, the wah and simple echo effects were run on the same signal in series (see Figure 7) to determine if the two effects were commutative. While I could not hear the difference, the fact that wah is not LTI could be sufficient to indicate that there is some difference between wah then echo and echo then wah. The difference is that because of the time varying nature of the wah band pass filter, the nature of what gets echoed depends upon the order of the two systems. Thinking about the specific example of a sine wave whose frequency changes linearly with time, in the wah then echo case, the signal is band-passed then echoed, and as a result, the echo will have 0.5 gain relative to the original (band-passed) signal. However, in the echo then wah case, the two components of the echoed signal will have frequency information due to the delay in the echo, and will thus not have the same gain when passing through the band-pass filter. This effectively demonstrates that the wah and echo operations are not commutative.

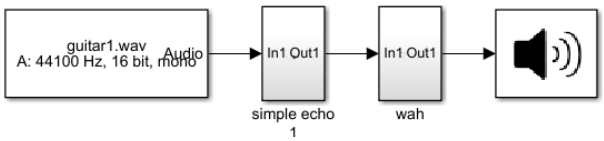

Figure 7: Simple echo and wah in series

3 Exercise B: Pitch Shifter

By following the instructions provided in the laboratory manual, two methods of pitch shifting were attempted (see the code in Section 4.2. Both resulted in a pitch shifted output that was not at all accelerated in time. The pitch was shifted up 2 half steps (1 whole step), as suggested in equation 4.

$$\frac{860 samples}{766 samples} \approx 1.227 = 2^{n/12} \therefore n = 12 \cdot log_2(1.227) \approx 2 \tag{4}$$

While both methods of pitch shifting resulted in an output with bad sound quality due to "clicks and pops," it is actually my opinion that the second method sounded worse than the first. The idea of using window weighted averages to smooth out the sound and remove clicks and pops is a valid one, and better preserves the integrity of the original audio signal, I found the effect on the output to be more negative than positive.

The artifacts (specifically distortion) are likely a result of discontinuities in the resampled signal. The window weighted averaging should reduce the presence of artifacts, but perhaps because we are only downsampling by a small factor (≈ 1.2), using the window actually creates more distortion but decreases the severity of the preexisting artifacts. Other methods of reducing distortion include (low pass) filtering and repeating the window smoothing method in smaller increments (i.e. instead of doing the same operation twice as many times and (weighted) averaging the results, doing it n (n>2) times, and (weighted) averaging the results). This window filtering actually acts as a low pass filter in some capacity to eliminate the high frequency artifacts created by the system (i.e. discontinuities in the output).

4 Matlab Code

4.1 Wah

```
function [num,den] = iirNotchLFO(LFO, f_min, f_max, Q)
Fs = 44100;
f = 0.5*(f_max-f_min)*double(LFO)+(f_min+f_max)/2; % Shift/Scale Triangle wave
BW = pi*f/(Fs*Q);
Wo = f*pi/Fs;
gain = 1/(1+sqrt(1/2).*tan(BW/2));
num = (1-gain)*[1 0 -1]';
den = [-2*gain*cos(Wo) (2*gain-1)]';
end
```

4.2 Exercise B: Pitch Shifter

```
% Lab 2
% Exerise B
[x, Fs] = audioread('singing44.wav');
```

```
p = 860; q = 766;
y = zeros(1, length(x)+1);
i = 0;
while i > -1
    for j = 0:q-1
        try
            y(q.*i + j + 1) = x(floor(q.*i + p.*j./q) + 1);
        catch
            i = -2;
        end
    end
    i = i+1;
end
%soundsc(x,Fs);
%pause
%soundsc(y, Fs);
%pause
m = 0:q-1;
w = sin(pi.*m./q).^2;
y2 = zeros(1, length(x)+1);
i = 0;
while i > -1
    for j = 0:floor(q/2)-1
        try
            y2(q.*i./2 + j + 1) = x(floor((q.*(i-1)/2)+((p.*(q./2)+j)/q))).*w((q/2) + j) + x(
        catch
            i = -2;
        end
    end
i = i + 1;
end
%soundsc(x,Fs);
%pause
soundsc(y2, Fs);
```